BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

BB

(12) UK Patent Application (19) GB (11) 2 283 595 (13) A

(43) Date of A Publication 10.05.1995

(21) Application No 9420379.1

(22) Date of Filing 10.10.1994

(30) Priority Data
(31) 08144388 (32) 02.11.1993 (33) US

(51) INT CL[6]
G06F 9/38 9/32

(52) UK CL (Edition N)
G4A APB

(56) Documents Cited
GB 2250840 A EP 0207665 A1 EP 0084114 A1

(58) Field of Search
UK CL (Edition M) G4A APB
INT CL[5] G06F 9/32 9/38
ONLINE DATABASES : WPI

(71) Applicant(s)
Motorola Inc

(Incorporated in USA - Delaware)

Corporate Offices, 1303 East Algonquin Road, Schaumburg, Illinois 60196, United States of America

(72) Inventor(s)
Paul C Rossbach
David S Levitan

(74) Agent and/or Address for Service
Peter D Hudson
Motorola Limited, European Intellectual Property Operation, Midpoint, Alencon Link, BASINGSTOKE, Hampshire, RG21 1PL, United Kingdom

(54) Data processor with both static and dynamic branch prediction, and method of operation

(57) A data processor (10) (Fig.1) has branch prediction circuitry 62 - 76 to predict a conditional branch instruction address before the condition on which the instruction is based is known. The branch prediction circuitry operates in one of two user selectable modes: dynamic branch prediction or static branch prediction. In the dynamic mode, the prediction is based upon a branch state history maintained at 62 for each branch instruction. Each branch state may be updated after the data processor determines if the prediction was correct. In the static mode, the branch prediction is based on one or more bits embedded in the branch instruction. The data processor may or may not update the branch state of each branch instruction during this second mode, as desired by the user. In either mode the predicted address is selected from the output of an incrementer 72 or a calculator 74, by a multiplexer 68.

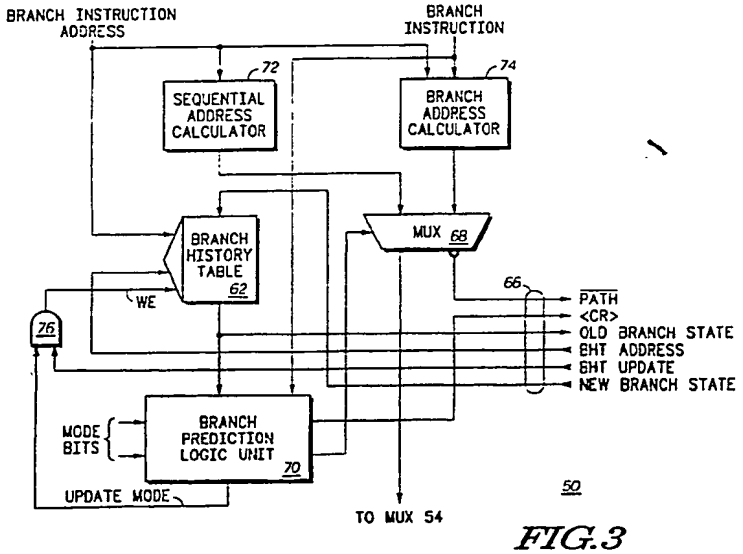

FIG.3

GB 2 283 595 A

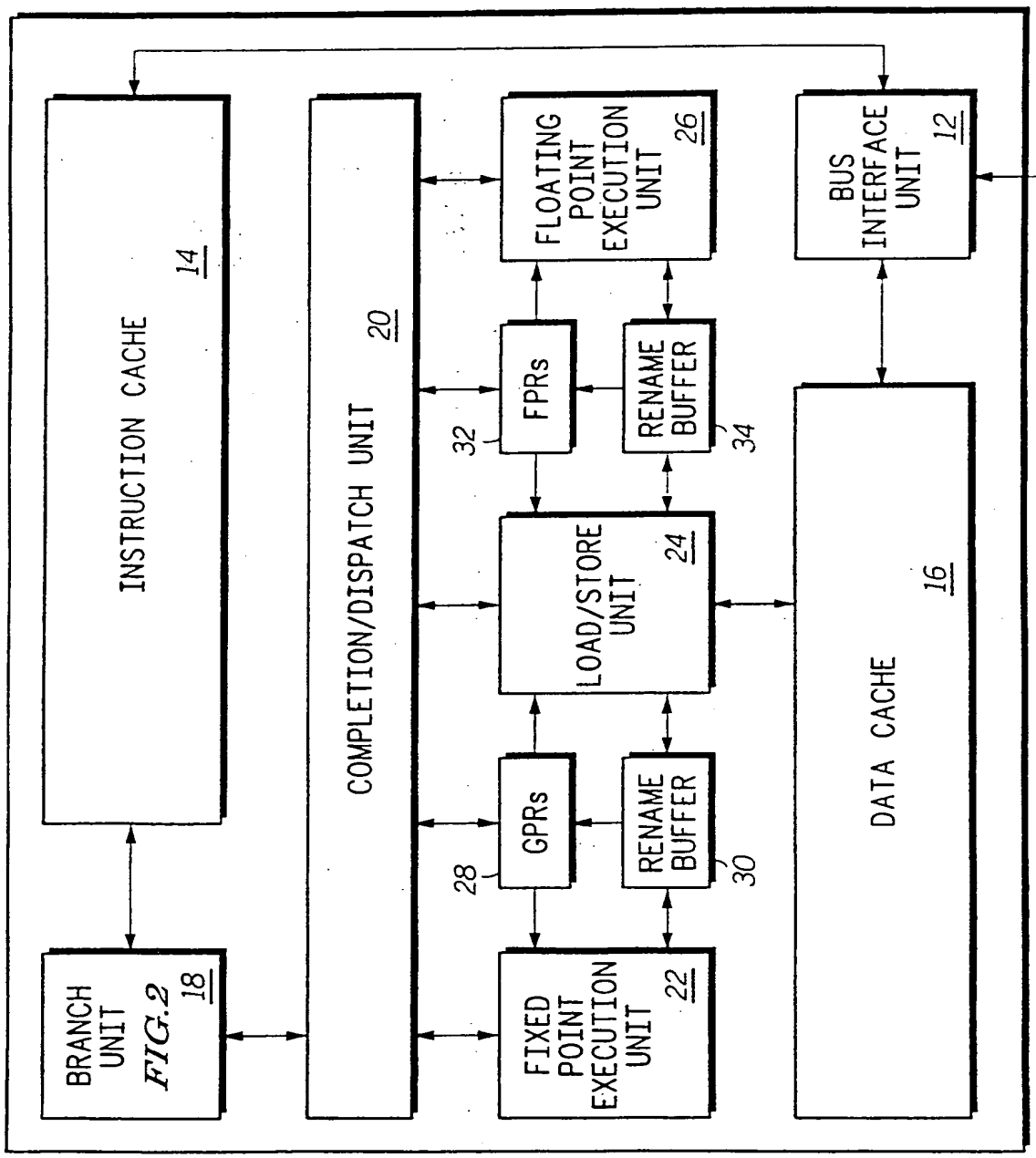

FIG.1

FIG.2

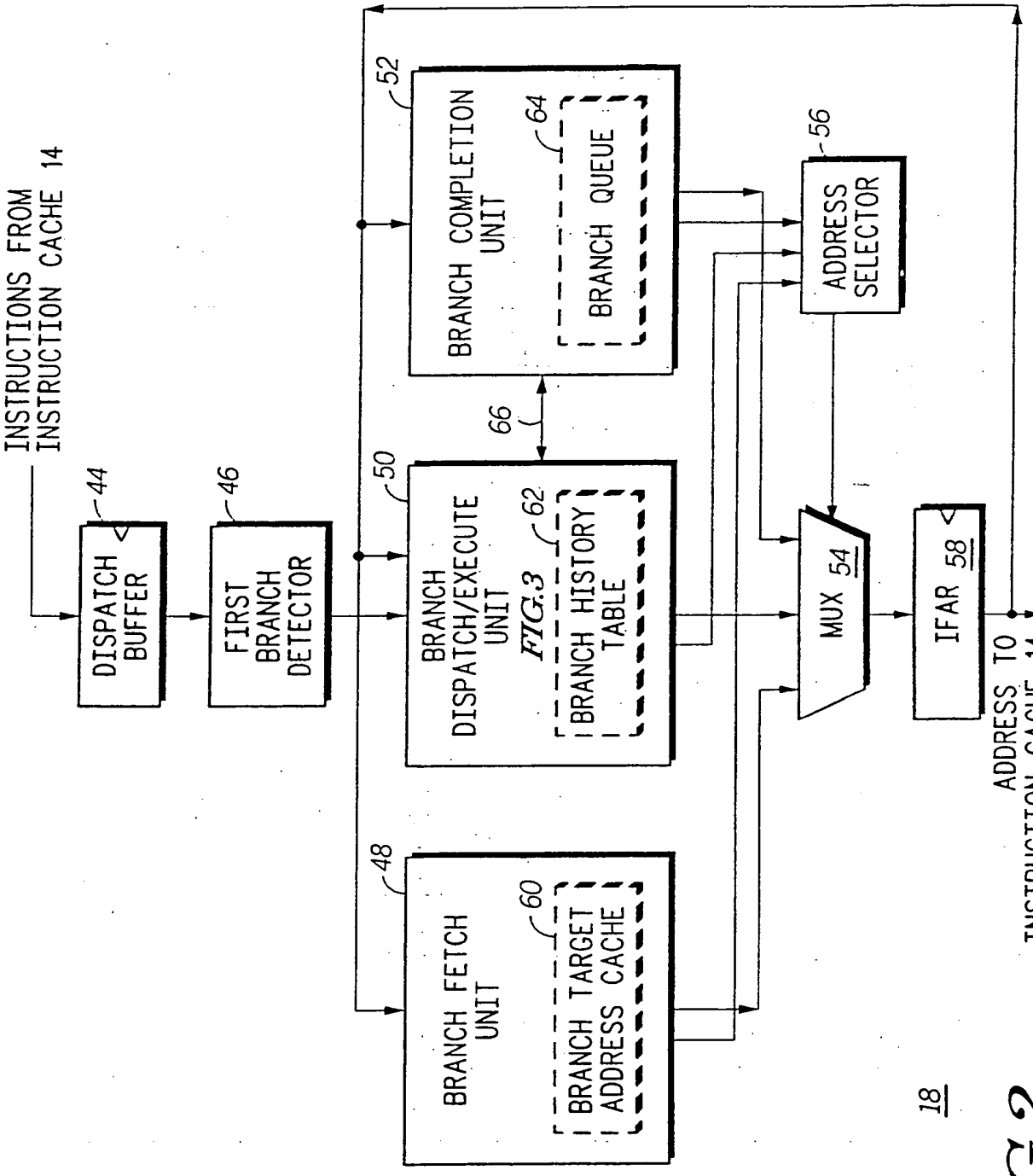

FIG.3

FIG.2

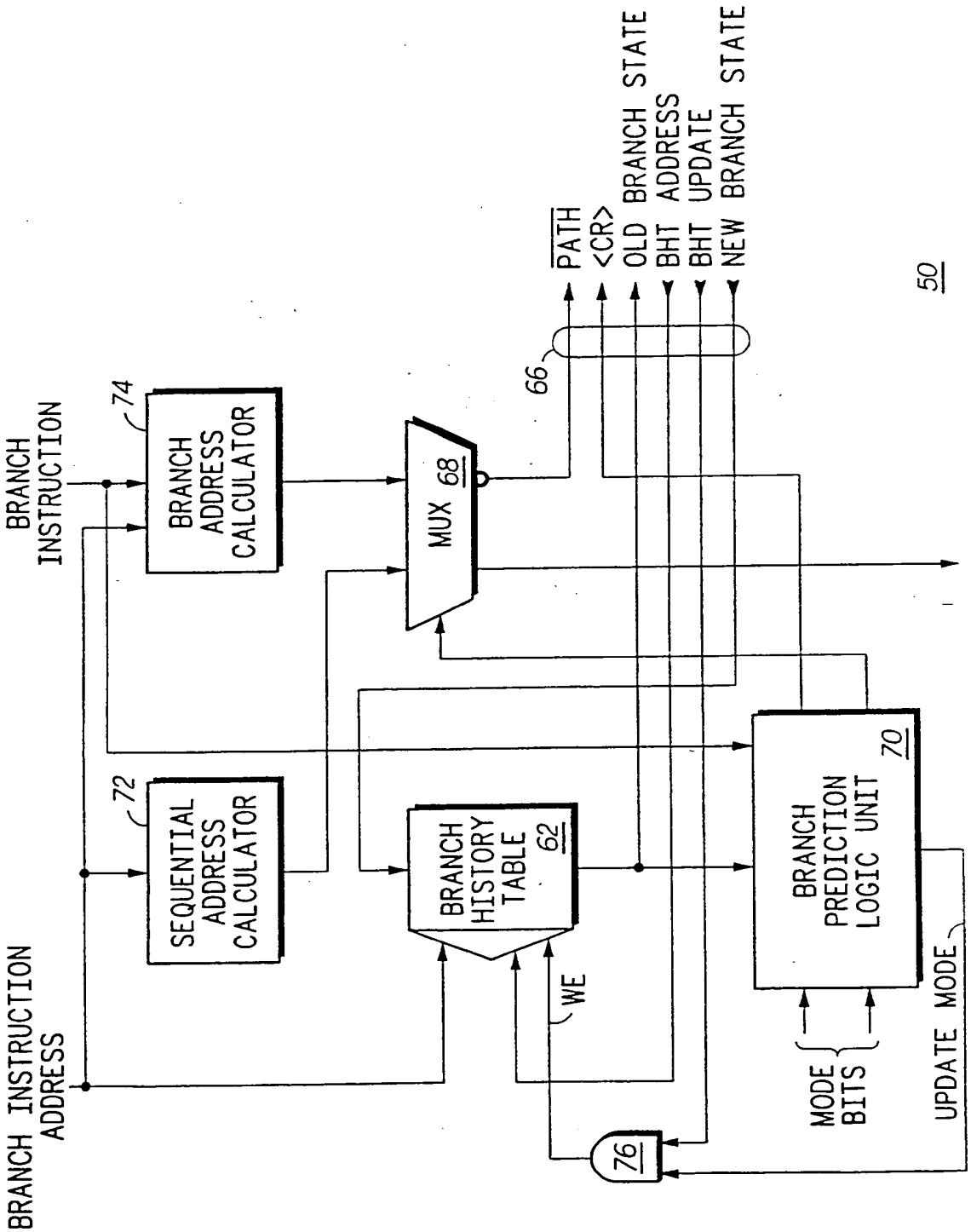

FIG.3

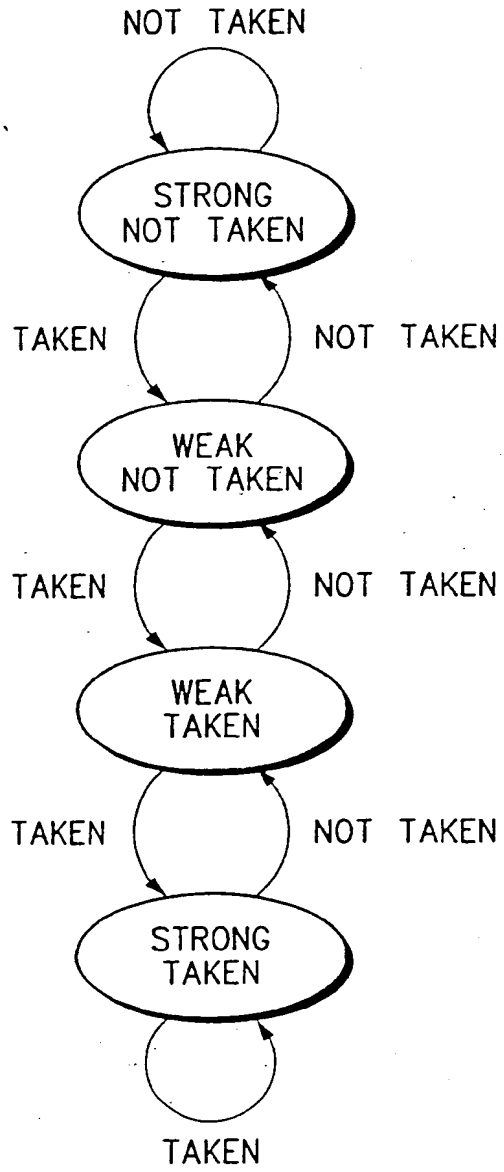

FIG.4

DATA PROCESSOR WITH BRANCH PREDICTION AND METHOD OF OPERATION

Field of the Invention

The present invention generally relates to digital computing systems, and more specifically to a data processor with branch prediction capabilities.

Background of the Invention

Branch prediction is one technique used to improve data processor performance. Data processors that use branch prediction techniques make a "guess" each time they receive a branch instruction, act on the guess, and then determine if the guess was correct by executing the instruction. Such a data processor guesses whether a branch will ultimately be taken and jump to a new instruction address or whether it will "fall through" to the next sequential instruction. Data processors that predict branch instructions gain performance because they can make an accurate guess faster than they can fully execute the branch instruction. These data processors then need only correct wrong guesses.

In general, data processors follow one of two branch prediction techniques when they make "guesses." First, a data processor may use a static branch prediction methodology. One or more bits in each branch instruction determine if each branch should be taken or not in a static branch prediction methodology. These bits are set once at program compilation and are, therefore, static. Second, a data processor may use a dynamic branch prediction methodology. In a dynamic branch prediction methodology, a state is defined for each branch instruction that reflects some past branch activity of the particular branch

instruction. The simplest state is branch-taken-last-time or branch-not-taken-last-time. The state for a particular branch instruction determines whether the branch should be taken or not taken the next time the data processor executes the particular branch instruction. The state of a particular branch instruction is modified, if necessary, after it completes to reflect whether the branch should have been taken or not taken.

Both branch prediction methodologies have advantages and disadvantages with respect to each other. For instance, static branch prediction may be advantageous in relatively short programs where the programmer's insight increases or at least maintains prediction accuracy relative to a dynamic branch prediction methodology. In these cases, a data processor incorporating dynamic branch prediction may not be able to accurately define the branch state for every possible branch instruction before the program completes. Conversely, dynamic branch prediction may be the only choice for predicting branch instructions in complex software programs. In these cases, the various branch states may be determined by a more sophisticated model. Such a model may increase the overall branch prediction accuracy of a data processor incorporating the methodology.

Summary of the Invention

In accordance with the present invention, there is disclosed a data processor having a branch prediction unit which substantially eliminates disadvantages of known data processors.

A data processor has first branch calculation circuitry, second branch calculation circuitry, first memory means, and branch prediction circuitry. The first and second branch calculation circuitries generate a first and a second fetch address, respectively, based on the address of an instruction and on the instruction itself, respectively. The first memory means stores a plurality of branch states associated with a differing one of a plurality of branch instructions. The branch prediction circuitry selectively outputs one of either the first or second fetch addresses responsive in a first mode of operation to the branch instruction, or responsive in a second mode of operation to one of the plurality of branch states.

A method of operating a data processor is also described comprising the steps of, at a first time, receiving a first branch instruction, generating a first fetch address, generating a second fetch address, selecting one of either the first or second fetch addresses responsive to the first branch instruction, and fetching instructions indexed by the selected fetch address. The method comprises the steps of, at a second time, receiving a second branch instruction, generating a third fetch address, generating a fourth fetch address, selecting one of either the third or fourth fetch addresses responsive to a branch state stored in a first memory means, and fetching instructions indexed by the selected fetch address.

Brief Description of the Drawings

The features and advantages of the present invention will be more clearly understood from the following detailed description taken in conjunction with the accompanying FIGURES where like numerals refer to like and corresponding parts and in which:

FIG. 1 depicts a block diagram of a data processor constructed in accordance with the present invention;

FIG. 2 depicts a block diagram of the branch unit depicted in FIG. 1;

FIG. 3 depicts a block diagram of the branch dispatch/execute unit depicted in FIG. 2; and

FIG. 4 depicts a state-transition diagram of an exemplary branch instruction.

Detailed Description of a Preferred Embodiment

FIG. 1 depicts a block diagram of a data processor 10 constructed in accordance with the present invention. Data processor 10 is a data processor that improves its performance by combing static and dynamic branch prediction methodologies. As described above, data processor 10 predicts whether each branch instruction will be taken or will not be taken. Data processor 10 then fetches instructions at the predicted address and begins executing these instructions. Later, data processor 10 resolves whether the branch should have been taken or should not have been taken and performs corrective measures if it predicted incorrectly at the earlier time. According to the disclosed invention, data processor 10 may operate in either a static or a dynamic branch prediction mode. Further, when in the static branch prediction mode, data processor 10 may or may not update the branch state of each branch instruction in a branch history table according to a programmable bit. The data processor may thereby "prime" its branch history table, for instance, when it executes program code for the first time.

Continuing with FIG. 1, a bus interface unit (hereafter BIU) 12 controls the flow of data between data processor 10 and the remainder of a data processing system (not depicted). BIU 12 is connected to an instruction cache 14 and to a data cache 16. Instruction cache 14 supplies an instruction stream to a branch unit 18 and to a completion/dispatch unit 20. Branch unit 18 is more fully described below in connection with FIG.s 2 and 3. Completion/dispatch unit 20 forwards individual instructions to an appropriate execution unit. Data processor 10 has a fixed point execution unit 22, a load/store execution unit 24, and a floating point execution unit 26. Fixed point execution unit 22 and load/store execution unit 24 read and write their results to a general purpose architectural register file 28, (labeled GPRs and

hereafter GPR file) and to a first rename buffer 30. Floating point execution unit 26 and load/store execution unit 24 read and write their results to a floating point architectural register file 32, (labeled FPRs and hereafter FPR file) and to a second rename buffer 34.

The operation of data processor 10 without the disclosed branch prediction methodology is known in the art. In general, branch unit 18 determines what sequence of programmed instructions is appropriate given the contents of certain data registers and the program steps themselves. Completion/dispatch unit 20 issues the individual instructions to the various execution units 22, 24 and 26. Each of the execution units performs one or more instructions of a particular class of instructions. The particular class of instructions of each execution unit is indicated by the name of the execution unit. For instance, floating point execution unit 26 executes floating point arithmetic instructions.

Fixed point execution unit 22 returns the results of its operations to designated entries in first rename buffer 30. First rename buffer 30 periodically updates an entry of GPR file 28 with an entry from first rename buffer 30 when all instructions preceding the instruction that generated the result have updated their GPR file entries. Completion/dispatch unit 20 coordinates this updating. Both first rename buffer 30 and GPR file 28 can supply operands to fixed point execution unit 22. Conversely, floating point execution unit 26 returns the results of its operations to designated entries in second rename buffer 34. Second rename buffer 34 periodically updates an entry of FPR file 32 with an entry in second rename buffer 34 when all instructions preceding the instruction that generated the result have updated their FPR file entries. Completion/dispatch unit 20 also coordinates this updating. Both second rename buffer 34 and FPR file 32 supply operands to floating point execution unit 26.

Load/store unit 24 reads data stored in GPR file 28, first rename buffer 30, FPR file 32 or second rename buffer 34 and

writes the selected data to data cache 16. This data may also be written to an external memory system (not depicted) depending upon operating characteristics of data processor 10 not relevant to the disclosed invention. Conversely, load/store unit 24 reads data stored in data cache 16 and writes the read data to GPR file 28, first rename buffer 30, FPR file 32 or second rename buffer 34.

The operation of data processor 10 with the disclosed branch prediction methodology is described below in connection with FIG.s 2 through 4. In general, data processor 10 is a reduced instruction set computer ("RISC"). Data processor 10 achieves high performance by breaking each instruction into a sequence of smaller steps, each of which may be overlapped in time with steps of other instructions. This performance strategy is known as "pipe lining."

In the depicted embodiment, each instruction is broken into as many as five discrete steps: fetch, dispatch, execute, write-back, and completion. Memory management circuitry (not shown) within instruction cache 14 retrieves one or more instructions beginning at a memory address identified by branch unit 18 during the fetch phase. Completion/dispatch unit 20 routes each instruction to the appropriate execution unit after determining that there are no impermissible data dependencies and after reserving a rename buffer entry for the result of the instruction in the dispatch phase. Each particular execution unit executes its programmed instruction during the execution phase and writes its result, if any, to the reserved rename buffer entry during the write-back phase. Finally, completion/dispatch unit 20 updates the architectural register files with the result of a particular instruction stored in a rename buffer after every instruction preceding the particular instruction has so updated the architectural register file. Generally, each instruction phase takes one machine clock cycle. However, some instructions require more than one clock cycle to execute while others do not require all five phases. There may also be a delay between the

write-back and completion phases of a particular instruction due to the range of times which the various instructions take to complete.

FIG. 2 depicts a block diagram of branch unit 18 depicted in FIG. 1. Branch unit 18 generates the address of the next instruction to fetch from memory (labeled ADDRESS TO MEMORY) and receives the next instruction fetched from instruction cache 14 (labeled INSTRUCTIONS FROM MEMORY). Branch unit 18 latches the received instructions in a dispatch buffer 44. In the depicted embodiment, branch unit 18 receives four instructions each clock cycle: the instruction residing at the forwarded address, the fetch address, and the three instructions residing at the three addresses following the fetch address. Therefore, a first branch detector 46 identifies the first branch instruction, if any, within each group of four latched instructions.

The address of the first instruction latched in dispatch buffer 44 is forwarded to a branch fetch unit 48, to a branch dispatch/execute unit 50 and to a branch completion unit 52. Branch fetch unit 48, branch dispatch/execute unit 50 and branch completion unit 52 each generate an address from which the next instruction should be fetched, though during different instruction phases. Branch dispatch/execute unit 50 also receives the output of first branch detector 46. The operation of these three units is described below. A multiplexer 54 (labeled MUX) outputs one of the three memory addresses responsive to an address selector 56. Address selector receives control signals from branch fetch unit 48, branch dispatch/execute unit 50 and branch completion unit 52, which it decodes to determine which of the three memory addresses it should output. An instruction fetch address register 58 (labeled and hereafter IFAR) latches the output of multiplexer 58. IFAR 58 generates the signal labeled ADDRESS TO MEMORY.

Branch fetch unit 48 calculates two new fetch addresses for a particular branch instruction as soon IFAR 58 latches a current fetch address. This calculation occurs during the particular branch instruction's fetch phase. The particular instruction(s)

indexed by the fetch address are not yet known during their fetch phase. Branch fetch unit 48 outputs one of these two addresses to multiplexer 54. The first address is output when the branch instruction associated with the current fetch address is assumed not taken or "falls through" to the next sequential instruction. The second address is output when the branch instruction associated with the current fetch address is assumed taken or the instruction stream "jumps" to a new location

Branch fetch unit 48 has two internal pipelines that each calculate one of these two new fetch addresses. The first branch fetch unit pipeline calculates a sequential address by incrementing the address latched by IFAR 58. In the described embodiment, this first branch fetch unit pipeline adds four instruction lengths to the contents of IFAR 58. The second branch fetch unit pipeline uses a portion of the contents of IFAR 58 to index a first block of random access memory (RAM) cache, called a branch target address cache 60 (hereafter BTAC). In the depicted embodiment, branch fetch unit 48 stores the branch target addresses (the fetch addresses) and a subset of the address of the branch instructions associated with the fetch addresses for a number of recent branch instructions in BTAC 60.

In the depicted embodiment, BTAC 60 is a two-way set associative cache. BTAC 60 uses a subset of the address latched in IFAR 58 to index two entries in BTAC 60. Branch fetch unit 60 then compares the remaining bits of the index address to a subset of address bits stored in each of the two indexed entries of BTAC 60. If one of the comparisons match, then a BTAC "hit" occurs and branch fetch unit 48 outputs the "taken" fetch address associated with the matching subset of address bits. If neither of the two comparisons match, then a BTAC "miss" occurs and branch fetch unit 60 outputs the incremented contents of IFAR 58, the "not taken" fetch address, to multiplexer 54.

Branch dispatch/execute unit 50 calculates two fetch addresses as soon as it receives the particular instructions indexed by the contents of IFAR 58 and first branch detector 46

identifies the first branch instruction, if any, in the four fetched instructions. Branch dispatch/execute unit 50 calculates the two addresses for a particular branch instruction during the particular instruction's dispatch phase. These two fetch addresses correspond to the branch-not-taken and branch-taken conditions. Branch dispatch/execute unit 50 outputs only one of these two addresses to multiplexer 54. It should be noted that branch fetch unit 48 has already output a new fetch address based on the same branch instruction to multiplexer 54 in the previous clock cycle. Depending upon the outcome of branch instructions that are simultaneously executing in either branch dispatch/execute unit 50 or branch completion unit 52, IFAR 58 may be loaded with the output of one of these two units. In this case, data processor 10 will be fetching instructions beginning at this address.

Branch dispatch/execute unit 50 also has two internal pipelines to calculate its two fetch addresses. The first pipeline calculates a sequential address by incrementing the branch instruction address by one instruction. First branch detector 46 calculates the first branch instruction address, if any, in every group of four fetched instructions The second branch dispatch/execute unit pipeline calculates the fetch address according to the particular branch instruction. For instance, this pipeline may add an offset embedded in the instruction to the branch instruction address.

Whether a branch instruction is taken or not is independent of where the branch jumps if it is taken. Conditional branch instructions are a class of branch instructions that depend upon an instruction result that is usually not known during the dispatch/execute phase of the branch instruction. This dependency determines if the branch is to be taken or not. It does not determine what the fetch address is. Therefore, branch dispatch/execute unit 50 selects one of its two addresses according to a dynamic branch prediction methodology or to a static branch prediction methodology. The specific prediction

methodology is user controllable by setting certain bits in a special purpose register (not shown).

While operating according to the dynamic branch prediction methodology, branch dispatch/execute unit 50 uses a portion of the contents of IFAR 58 to index a second block of RAM called a branch history table 62 (hereafter BHT). Branch dispatch/execute unit 50 defines a branch state for each branch instruction or for some subset of branch instructions. The state of a particular branch instruction determines if the branch is taken or not taken. Branch dispatch/execute unit 50 updates the branch state of a particular branch instruction after the instruction completes. The branch state model for the depicted embodiment is described below in connection with FIG. 4.

While operating according to the static branch prediction methodology, branch dispatch/execute unit 50 uses one or more bits embedded in the branch instruction itself to determine if the branch is taken or not. These bits are set once when the program containing the branch instruction is compiled.

Branch dispatch/execute unit 50 may or may not update the update BHT 62 while it is operating according to the static branch prediction methodology. The decision to update BHT 62 or not to update BHT 62 may be a strategic decision. For instance, data processor 10 may operate in the static-branch-prediction-with-BHT-update-mode after beginning a program. Data processor 10 might then switch to dynamic-branch-prediction-mode once a certain number of branch instructions have executed and BHT 62 is populated with data. Also, when switching from a first to a second routine and back again, it may be advantageous to switch from dynamic branch prediction to static branch prediction with no update and back again to avoid contaminating BHT 62 with data associated with the second routine if the second routine is short or otherwise less important than the first routine.

Branch completion unit 52 generates a single address for a particular branch instruction after the particular branch instruction is completed. A branch instruction, like any other

instruction, is completed when all instructions preceding the branch instruction have written their results to the appropriate architectural register. By this time, the condition on which the branch instruction is based and, hence, whether the branch should be taken or should be not taken is known. It should be noted that branch fetch unit 48 has already output an instruction fetch address based on the same branch instruction to multiplexer 54 in the fetch phase of the same branch instruction and that branch dispatch/execute unit 50 may have output a second instruction fetch address to multiplexer 54 in the dispatch/execute phase of the same instruction.

Branch completion unit 52 has within it a first-in-first-out queue, or a branch queue 64 in which it stores data about each executed branch instruction. Branch completion unit 52 receives certain data about each instruction from branch dispatch/execute unit 55 through a set of data/control signals 66: (1) the fetch address generated by branch dispatch/execute unit 50 but not output to multiplexer 54 (the "not selected path"); (2) the predicted value of the condition on which the branch was calculated during the dispatch/execute phase; and (3) the branch state of the branch instruction. Branch completion unit 52 also stores the address of each branch instruction in branch queue 64. Branch completion unit 52 receives a control signal from completion/dispatch unit 20 (not depicted) indicating when each branch instruction completes.

After a branch instruction completes, branch completion unit 52 receives the actual value of the branch condition on which the dispatch/execute fetch address was based. Typically this condition is a bit in a special purpose register (a condition register) that may be modified by other instructions. If the actual value of the condition differs from the predicted value stored in branch queue 64, then the branch prediction made by branch dispatch/execute unit 50 was wrong. In this case, branch completion unit 52 outputs the stored address associated with the branch instruction. This address is the path that was originally

not selected by branch dispatch/execute unit 50. If the actual value of the condition is the same as the predicted value stored in branch queue 64, then the branch prediction made by branch dispatch/execute unit 50 was correct. In this case, branch completion unit 52 does not need to do anything except invalidate the entry in branch queue 64 associated with the complete branch instruction for future use. In either case, branch completion unit 52 generates a new branch state for the branch instruction based on the comparison of the predicted and actual branch conditions and on the stored branch state. Branch completion unit 52 forwards the new branch state to BHT 62 using the stored branch address as an index to BHT 62.

Address selector 56 determines which of up to three output addresses it should cause multiplexer 54 to output to IFAR 58. Address selector 56 receives the fetch address output from branch fetch unit 48, the fetch address output from branch dispatch/execute unit 50 and a control signal from branch completion unit 52. Branch completion unit 52 asserts this control signal if the predicted value of the condition and the actual value of the condition differ. If branch completion unit 52 asserts its control signal, then address selector 56 causes multiplexer 54 to output the address generated by branch completion unit 52. If branch completion unit 52 does not assert its control signal and the address generated by branch dispatch/execute unit 50 differs from the address generated by branch fetch unit 48 during the previous clock cycle, then address selector 56 causes multiplexer 54 to output the address generated by branch dispatch/execute unit 50. Otherwise, address selector 56 causes multiplexer 54 to output the address generated by branch fetch unit 48. It should be understood that address selector 56 may be selecting one of up to three different addresses generated by three different branch instructions at any given moment.

FIG. 3 depicts a block diagram of branch dispatch/execute unit 50 depicted in FIG. 2. Branch dispatch/execute unit 50 has a

multiplexer 68 (labeled MUX) that outputs one of two addresses to multiplexer 54 (FIG. 2) via the path TO MUX 54. A branch prediction logic unit 70 selects which of the two addresses multiplexer 68 will output. Branch prediction logic unit 70 is described below.

Multiplexer 68 receives a first address from sequential address calculator 72. Sequential address calculator receives the branch instruction address during the dispatch phase and increments the binary number by one instruction. Sequential address calculator 72 generates the address of the next instruction to fetch from memory when the branch instruction is predicted not to be taken.

Multiplexer 68 receives a second address from branch address calculator 74. Branch address calculator 74 generates a fetch address according to a received branch instruction and to a received branch instruction address. For instance, branch address calculator may add an offset embedded in the instruction to or subtract it from the branch instruction address or may simply pass an absolute address embedded within the branch instruction to multiplexer 68. Branch address calculator 74 generates the address of the next instruction to fetch from memory when the branch instruction is predicted to be taken.

Branch prediction logic unit 70 operates in one of three modes responsive to the input signals MODE BITS: (1) dynamic branch prediction; (2) static-branch-prediction-with-BHT-update; and (3) static-branch-prediction-without-BHT-update. In the preferred embodiment, the input signals MODE BITS are forwarded from a special purpose register (not shown) which is user programmable.

In the dynamic branch prediction mode, the address stored in IFAR 58 is used to index BHT 62. BHT 62 outputs one or more bits that define a branch state for an associated branch instruction. In the depicted embodiment, each branch instruction is in one of four branch states. If the branch instruction is defined by either of two first states, then branch prediction logic unit 70 will select

the address generated by sequential address calculator 72. If the branch instruction is defined by either of two second states, then branch prediction logic unit 70 will select the address generated by branch address calculator 74. The branch states are described below in connection with FIG. 4. Branch prediction logic unit 70 asserts a control signal UPDATE MODE in this mode. An AND gate 76 receives the control signals UPDATE MODE and BHT UPDATE from branch prediction logic unit 70 and branch completion logic unit 52, respectively. An output of AND gate 72 generates a BHT 62 write-enable signal (labeled WE). When both UPDATE MODE and BHT UPDATE are asserted, branch completion unit 64 may write a new branch state to BHT 62. Branch completion unit 64 writes a new branch state to BHT 62 during the completion phase of the branch instruction by asserting the control signal BHT UPDATE and by driving the branch address and new branch state onto the data paths BHT ADDRESS and NEW BRANCH STATE, respectively.

In the static-branch-prediction-with-BHT-update mode, branch prediction logic unit 70 receives one or more bits from the branch instruction itself via the data path BRANCH INSTRUCTION. In the depicted embodiment, two bits are used to predict branch instructions in the static branch prediction mode. One of these bits is a sign bit indicating, generally, whether branch address calculator 74 adds two numbers to generate a fetch address or subtracts two numbers to generate the fetch address (a forward branch or a backward branch). The second bit is a branch prediction bit that indicates whether the branch should be taken or not. The relationship between the logic state of the branch prediction bit and the resulting action reverses when the sign bit is toggled. Regardless, the sign bit and the branch prediction bit decode to a single taken/not-taken bit. If the single taken/not-taken bit is logically equivalent to a first state, then branch prediction logic unit 70 will select the address generated by sequential address calculator 72. If the single taken/not-taken bit is logically equivalent to a second state, then branch prediction logic unit 70 will select the address generated by

branch address calculator 74. Branch prediction logic unit 70 also asserts the control signal UPDATE MODE in this mode.

In the static-branch-prediction-without-BHT-update mode, branch prediction logic unit 70 operates identically as in the static-branch-prediction-with-BHT-update mode with one exception. In this mode, branch prediction logic unit 70 does not assert the control signal UPDATE MODE. Therefore, branch completion unit 52 can not update BHT 62.

Branch dispatch/execute unit 50 has certain operating characteristics common to all three branch prediction modes. Multiplexer 68 outputs the unselected fetch address to the data path $\overline{\text{PATH}}$. Branch prediction logic unit 70 outputs the assumed value of the branch condition to the data path ⟨CR⟩. BHT 62 outputs the branch state of the branch instruction to the data path OLD BRANCH STATE. As described above, the unselected fetch address, the assumed value of the branch condition, and the branch state of the branch instruction are saved in branch queue 52 along with the address of the branch instruction until the branch instruction completes. At completion, branch completion unit 52 determines if branch dispatch/execution unit 50 made a correct prediction. At that time, the stored data values are used to generate a corrected fetch address and a new branch state for BHT 62, if necessary.

FIG. 4 depicts a state-transition diagram of an exemplary branch instruction. Each branch instruction or each instruction of a subset of the branch instructions has a branch state defined for it. These branch states are stored in BHT 62 and indexed by the address of the branch instruction. The branch state of each branch instruction may be in one and only one of four states (labeled and hereafter STRONG-NOT-TAKEN, WEAK-NOT-TAKEN, WEAK-TAKEN and STRONG-TAKEN). On reset of data processor 10, each branch state is placed in a predetermined branch state. Otherwise, branch completion unit 52 modifies the branch state of a particular branch instruction as the particular branch instruction completes.

Branch dispatch/execute unit 50 predicts that a branch will not be taken if the instruction's branch state corresponds to STRONG-NOT-TAKEN or WEAK-NOT-TAKEN. Branch dispatch/execute unit 50 predicts that a branch will be taken if the instruction's branch state corresponds to STRONG-TAKEN or WEAK-TAKEN. As described above, branch completion unit 52 will change the state of a branch instruction after the branch instruction completes.

If the branch state of an instruction corresponds to STRONG-NOT-TAKEN and the branch was resolved as taken at instruction completion, then branch completion unit 52 will change the branch state to WEAK-NOT-TAKEN. A branch instruction is resolved as taken at instruction completion if the predicted branch condition and actual branch condition are not equal in the STRONG-NOT-TAKEN and WEAK-NOT-TAKEN states. Otherwise, branch completion unit 52 will write the same branch state, STRONG-NOT-TAKEN, to the BHT entry indexed by the branch address.

If the branch state of an instruction corresponds to WEAK-NOT-TAKEN and the branch was resolved as taken at instruction completion, then branch completion unit 52 will change the branch state to WEAK-TAKEN. However, if the branch state of an instruction corresponds to WEAK-NOT-TAKEN and the branch was resolved as not-taken at instruction completion, then branch completion unit 52 will change the branch state to STRONG-NOT-TAKEN. A branch instruction is resolved as not taken at instruction completion if the predicted branch condition and actual branch condition are equal in the STRONG-NOT-TAKEN and WEAK-NOT-TAKEN states.

If the branch state of an instruction corresponds to WEAK-TAKEN and the branch was resolved as taken at instruction completion, then branch completion unit 52 will change the branch state to STRONG-TAKEN. A branch instruction is resolved as taken at instruction completion if the predicted branch condition and actual branch condition are equal in the WEAK-TAKEN and STRONG-TAKEN states. However, if the branch state of an instruction

corresponds to WEAK-TAKEN and the branch was resolved as not-taken at instruction completion, then branch completion unit 52 will change the branch state to WEAK-NOT-TAKEN. A branch instruction is resolved as taken at instruction completion if the predicted branch condition and actual branch condition are not equal in the WEAK-TAKEN and STRONG-TAKEN states.

If the branch state of an instruction corresponds to STRONG-TAKEN and the branch was resolved as not-taken at instruction completion, then branch completion unit 52 will change the branch state to WEAK-TAKEN. Otherwise, branch completion unit 52 will write the same branch state, STRONG-TAKEN, to the BHT entry indexed by the branch address.

Although the present invention has been described with reference to a specific embodiment, further modifications and improvements will occur to those skilled in the art. For instance, the disclosed invention may be incorporated into data processors traditionally classified as complex instruction set computers or CISC machines. Also, certain functional units may be omitted in certain embodiments or relocated to other areas of data processor 10. It is to be understood therefore, that the invention encompasses all such modifications that do not depart from the spirit and scope of the invention as defined in the appended claims.

Claims

What is claimed is:

1. A data processor with branch prediction comprising:
   first branch calculation circuitry generating a first fetch address responsive to an address of a branch instruction;
   second branch calculation circuitry generating a second fetch address responsive to the branch instruction, a result of the branch instruction predicated on a condition;
   first memory means storing a plurality of branch states, each of the plurality of branch states associated with a differing one of a plurality of branch instructions including the branch instruction; and
   branch prediction circuitry coupled to the first and second branch calculation circuitries and to the first memory means, the branch prediction circuitry selectively outputting one of either the first or second fetch addresses responsive in a first mode of operation to the branch instruction, responsive in a second mode of operation to one of the plurality of branch states.

2. The data processor of claim 1 further comprising branch completion circuitry coupled to the first memory means and to the branch prediction circuitry, the branch completion circuitry outputting a third fetch address responsive to receiving the condition, the third fetch address logically equivalent to the one of either the first or second fetch address not output by the branch prediction circuitry.

3. The data processor of claim 2 wherein the branch completion circuitry comprises circuitry generating the plurality of branch states, each one of the plurality of branch states responsive to a previously stored one of the plurality of branch states and to the condition, and wherein the branch prediction circuitry further comprises circuitry storing the plurality of branch states in the first memory means in a third mode and not storing the plurality of branch states in a fourth mode.

4. The data processor of claim 3 further comprising:

third branch calculation circuitry generating a fourth fetch address responsive to the address of the first branch instruction;

second memory means outputting one of either one or none of a plurality of addresses responsive to an input address, each one of the plurality of addresses associated with a differing one of a plurality of address indices, the input address equivalent to a portion of the address of the first branch instruction; and

output circuitry coupled to the third branch calculation circuitry and to the second memory means, the output circuitry outputting the fourth fetch address responsive to a second memory means miss, the output circuitry outputting the one of the either one or none of a plurality of addresses responsive to a second memory means hit.

5. The data processor of claim 2 further comprising:

third branch calculation circuitry generating a fourth fetch address responsive to the address of the first branch instruction;

second memory means outputting one of either one or none of a plurality of addresses responsive to an input address, each one of the plurality of addresses associated with a differing one of a plurality of address indices, the input address equivalent to a portion of the address of the first branch instruction; and

output circuitry coupled to the third branch calculation circuitry and to the second memory means, the output circuitry outputting the third fourth address responsive to a second memory means miss, the output circuitry outputting the one of the either one or none of a plurality of addresses responsive to a second memory means hit.

6. A method of operating a data processor comprising the steps of:

at a first time period, receiving a first branch instruction in a first and a second branch calculation circuitry of the data processor;

generating a first fetch address responsive to an address of the first branch instruction in the first branch calculation circuitry;

generating a second fetch address responsive to the first branch instruction in the second branch calculation circuitry;

selecting, by a branch prediction circuitry coupled to the first and second branch calculation circuitries, one of either the first or second fetch addresses responsive to the first branch instruction;

fetching instructions indexed by the selected one of either the first or second fetch address;

at a second time period, receiving a second branch instruction in the first and the second branch calculation circuitry;

generating a third fetch address responsive to an address of the second branch instruction in the first branch calculation circuitry;

generating a fourth fetch address responsive to the second branch instruction in the second branch calculation circuitry;

selecting, by the branch prediction circuitry, one of either the third or fourth fetch addresses responsive to a first selected one of a plurality of branch states, the plurality of branch states stored in a first memory means, each of the plurality of branch states associated with a differing one of a plurality of branch instructions including the second branch instruction; and

fetching instructions indexed by the selected one of either the third or fourth fetch address.

7. The method of claim 6 wherein the step of selecting one of either the third or fourth fetch addresses occurs prior to the step of selecting one of either the first or second fetch addresses.

8. The method of claim 7 wherein the step of selecting one of either the third or fourth fetch addresses further comprises the steps of:

generating a first branch state; and

storing the first branch state in the first memory means.

9. The method of claim 8 further comprising the steps of:

at a third time period subsequent to the first time, receiving a third branch instruction in the first and the second branch calculation circuitry;

generating a fifth fetch address responsive to an address of the third branch instruction in the first branch calculation circuitry;

generating a sixth fetch address responsive to the third branch instruction in the second branch calculation circuitry; and

selecting one of either the fifth or sixth fetch addresses responsive to a second selected one of the plurality of branch states;

fetching instructions indexed by the selected one of either the fifth or sixth fetch address;

generating a second branch state; and

storing the second branch state in the first memory means.

10. The method of claim 9 wherein the step of selecting one of either the first or second fetch addresses further comprises the steps of:

generating a branch state; and

storing the branch state in the first memory means.

11. The method of claim 10 further comprising the steps of:

at a fourth time period prior to the first time, generating a seventh fetch address responsive to the address of the first branch instruction in a fifth branch calculation circuitry;

outputting one of a plurality of addresses and one of a plurality of address tags from a third memory means responsive to the address of the first branch instruction;

selecting one of either the seventh fetch address or the one of the plurality of addresses responsive to a logical comparison of the one of the plurality of address tags and the address of the first branch instruction; and

fetching instructions indexed by the selected one of either the seventh fetch address or the one of the plurality of addresses.

12. The method of claim 8 wherein the step of selecting one of either the first or second fetch addresses further comprises the steps of:

generating a branch state; and

storing the branch state in a first memory means.

13. The method of claim 6 wherein the step of selecting one of either the first or second fetch addresses further comprises the steps of:

generating a branch state; and

storing the branch state in a first memory means.

Patents Act 1977
Examiner's report to the Comptroller under Section 17
(The Search report)

96

Application number
GB 9420379.1

Relevant Technical Fields

(i) UK Cl (Ed.M) G4A (APB)

(ii) Int Cl (Ed.5) G06F (9/32, 9/38)

Search Examiner
B G WESTERN

Date of completion of Search
23 NOVEMBER 1994

Databases (see below)
(i) UK Patent Office collections of GB, EP, WO and US patent specifications.

(ii) ON-LINE DATABASE; WPI

Documents considered relevant following a search in respect of Claims :-
1 TO 13

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family; corresponding document.

Category	Identity of document and relevant passages		Relevant to claim(s)
A	GB 2250840 A	(INTEL) see whole document	
A	EP 0207665 A1	(HEWLETT-PACKARD) see whole document	-
A	EP 0084114 A1	(IBM) see whole document	

Databases: The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journal (Patents). The on-line databases considered for search are also listed periodically in the Official Journal (Patents).